MBTA Subway Sim

Max Mitchell

Tufts University

https://github.com/maxmmitchell

max.m.mitchell@gmail.com

Acknowledgments

I'd like to express my gratitude to a number of people who helped me on this project. To Richard Townsend, my advisor, who provided lots of support, encouragement, and feedback from the beginning through the end. To Liping Liu and Shan Jiang, for providing advice in a field which was new to me. To my brother, who pushed me to try something new. And finally, to all my friends and family, for everything.

I. Introduction

My project is a simulation of the MBTA transit system. It leverages entry/exit data to build a model of possible rider behavior: which stops people who are riding the T are coming from/going to, at what time of day. This model has been designed for use in a simulation, where users can modify the subway train routes. The simulation then calculates the expected impact on current users of the T. This simulation enables data visualization of how these changes might be utilized by current passengers, affect a passenger's average ride time, load on the MBTA, and more.

I made this because I have been fascinated by the public transit system in Boston. I thought this project would not only build skills in handling APIs and large datasets, but also grow my understanding of problem solving in a more general context, when there is more design involved and less of a right/wrong answer. Additionally, user interaction has always been an element of programming projects I have cared deeply about; from the video game coding that spawned my interest in CS many years ago, to the grammar, usability, and documentation of programming languages that became a key focus of mine when writing a compiler in my final semester of undergrad.

Another reason I made this is simply because it doesn't exist in a user-friendly format. I've found a great visualizer, but it's from 2014, and only examines the red, blue, and orange lines. TransitMatters does an awesome job of documenting speeds, slowdowns, passenger use, accessibility concerns, and more, but they're mostly focused on documenting performance. There's a myriad of apps which directly patch users to the MBTA data on bus/train locations, for planning trips. While there is research extant in this field of applying data-driven models to predict behavior like this, I wish to apply the research of others to create models specifically for the MBTA, and with an interactable feature.

This document has served as a living log of my progress as I moved through the project over the past year. It tracks the issues I ran into, how I solved them, and ultimately what my results were. I'd like to note that much of this document describes both rail and bus data; in the end, I decided to omit bus data from my model and simulation. It was getting too complicated, and I was overworked. I have left in the details regarding the bus data simply as a record of the work I did with the bus data before deciding to scrap it.

The source code and data for this project, in its entirety, can be found on GitHub at https://github.com/maxmmitchell/mbta-subway-sim. It is available for forking, cloning, and contains a README with all relevant information to get started with the project. Later in this document, I discuss Future Work. I encourage any interested readers to approach any of my suggested ideas, or devise your own.

II. Data

I've had some data limitations. I was able to get a hold of the entry/exit data I wanted for buses and trains, but it had been substantially aggregated. For each stop, I only have three days: 'weekday', 'saturday', and 'sunday'. For the weekday, I get time in blocks (e.g. early am, peak am, etc.). For the weekends, I get no time blocks, making the data more or less useless for me. Instead of getting a year's worth, for the years 2017-2019 (2016-2022 for buses) I get the 'Fall' data aggregated onto one weekday, giving me a total and average for entries/exits across all weekdays. Obviously, this is subpar. I'd been in contact with some folks at MBTA about getting better data, but by the time they sent me what I wanted, it was too late to integrate it into the model. The "better" data was useful for testing though. More on that later. I decided to work with 2019 data for consistency.

This project has required extensive data manipulation to get the data that I can get my hands on into forms that are useful for what I'm trying to do. Fortunately, python has the pandas library which has been a great help. One of the more complicated difficulties was reckoning with stops that have been removed or changed since the ridership data I have was collected. These stops have been left out of the dataset on stop coordinates, making them harder to map. For the various changes, I've had to manually edit my data on stop coordinates.

For the subway, from 2019 to 2023, it appears there were three stops extant in 2019 ridership data which disappeared in the 2023 location data: Boston University West (place-buwst), Saint Paul Street (place-stplb), and Pleasant Street (place-plsgr). Saint Paul Street is tricky, because it appears there were **two** Boston stops called Saint Paul Street: place-stpul, at the intersection of Beacon Street and Saint Paul Street, which still exists on GLC today, and place-stplb, which was on GLB, sandwiched between place-buwst and place-plsgr. place-buwst and place-stplb have been rolled into the Amory Street station today on GLC, whereas place-plsgr has been rolled into Babcock Street station. Initially, I added appropriate records pointing to the new stations which replaced the old ones. For the bus, there are nearly 600 stops which have ridership records from 2019 that didn't appear in the 2023 dataset of stop locations. This was simply too much. Fortunately, I found archival GTFS data on MBTA's website which allowed me to find stop data for bus and subway from 2019, which appeared to work much better. It required minimal tweaking around a few choice bus stops. First, a few stops on Saratoga St on the 712/713 bus routes appear to have been merged into one – Saratoga St @ Teregram St. This was easy enough to fix. Second, there were five stops relating to the Ted Williams Tunnel on the SL1 and SL3 routes which no longer exist. Seems like at one point, a rider could enter/exit closer to the tunnel. They were remapped to the nearest stop based on their directionality and route. This process would need to be repeated for any years other than 2019 which I wish to use.

III. Architecture

A. The Map

Originally, my architecture was dependent on the Google Maps API for calculating the distance between two stops, time-wise. For the model and simulation, it was critical to be able to determine the time between two stops. Google Maps API, however, is neither cheap nor fast enough for the amount of calls I was making. I devised to create my own map of the MBTA subway. I built a JSON file to track the layout of the stops, and timing between their neighbors, and wrote a simple algorithm to use that information to pre-compute the time for all possible rides. This was a great improvement to the efficiency of the model and simulation, although it came at a cost of some tedium and accuracy.

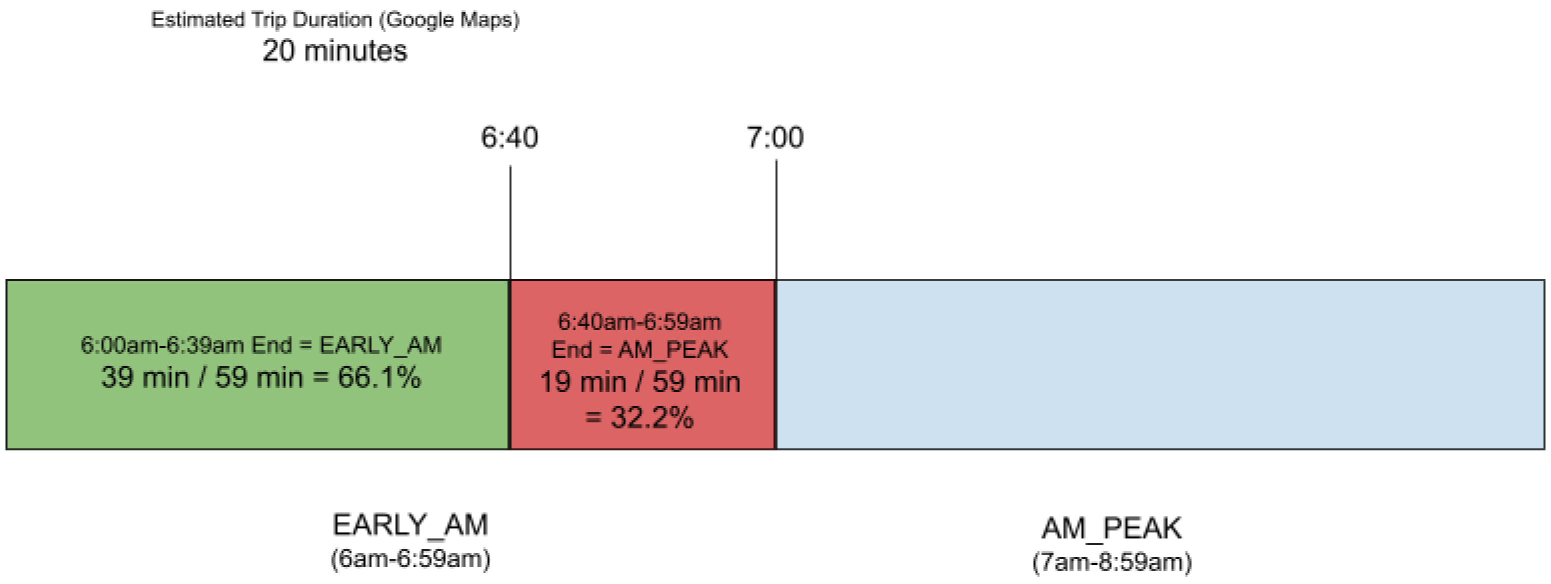
I'll note that I've chosen a pattern of my own for what "inbound" and "outbound" mean. The whole concept of inbound vs. outbound is somewhat nonsensical and mapping-wise makes no sense. A train goes in the same direction on the rail for the whole duration between its origin and destination, yet at some point in Boston, the train coming from Alewife shifts from an inbound to an outbound train, because it's closer to its destination than its origin? Balderdash I say! The direction is the same! I refuse to play their stupid game! The pattern I've chosen is that north of the river is outbound, south of river is inbound. Consistency is quite valuable in my map's design.

I'd also like to note that my map contains no Mattapan or Green Line Extension, for simplicity of use with older data. Furthermore, I've divided the Green Line in a way such that my map can have single transfers for computation purposes…this means I misrepresent where the trains actually go, but it doesn't really matter since transfers are free on my map. I discuss this in more detail in my README on GitHub, as it's more relevant when actually using the code. To avoid repeating myself, I'll direct the reader there if they are curious about further map details.

B. The Model

For my model's architecture, I decided to use a genetic algorithm. Upon further review of the literature I presented in my original design document, I discovered that my goals and datasets were incompatible with the Markov models presented for predicting passenger behavior. Instead, I implemented the following genetic algorithm structure:

1. Generate many individuals, each representing a day's worth of passengers (default for a weekday is ~671,000 for rail in 2019, and 77,300 for bus). This is a set of pseudo-random paths across the city, serviceable by public transit. Each of these paths have the following data, or chromosomes:
    a. Start *time* (which should be within the range of MBTA operating hours)
        i. Generated pseudo-randomly, with a bias towards times with higher ridership.
        ii. Due to limitations on the MBTA Blue Book data, I'm currently using time blocks (e.g. early AM, midday, etc.)
    b. Start/End Stop
        i. Generated pseudo-randomly, with a bias towards stops with higher ridership during the pre-determined start time block.
    c. End time
        i. I used my homebrewed map to calculate the estimated end time of the trip. Then, calculate whether we will consider the trip to end in the current time block or the next. Consider the following schema for such a calculation:



This is imperfect, but since I'm working with time blocks anyway, and just trying to get a sense for whether the trip would more likely end in the current time block or the next time block, I think it is ok.

2. Calculate fitness of every individual with the following algorithm:
    a. Calculate the number of ons/offs at each time of day for each stop across all rides in an individual. This will create a dataset which is of the same measurements made by the MBTA for ridership.
    b. Compare with our existing data. We calculate fitness using a normal distribution[1]. Determine the number of standard deviations from the mean for each station/time-of-day combination, and average.
3. Eliminate weakest individuals, and generate fresh individuals with mutation, crossover, fresh generation, etc. Tweak as needed.
4. Repeat until fitness plateaus/similarity reaches desirable level for top individuals.

Once the model is substantially trained, the simulating can begin.

C. The Simulation

My simulation predicts the impacts to ridership of adding/subtracting stations. But before getting into that, I must note how more granular locations can assist in the simulating process by allowing more decisive re-routing. It is useful to the simulation to assign each ride a destination and origin beyond just the stop, to the extent that it can impact how a rider might decide to use a new stop instead of their current choice, or how they might reroute when their selected stop is removed.

Ideally, I could assign these riders actual destinations, but in order to generate this, I would need to construct a list of all buildings serviceable by the MBTA. This would be difficult; maybe Google Maps API could help me out here, but it's beyond the scope of this project; see **Future Work** for more discussion of this. Here is the current process for developing granularity beyond the station:

1. Consider a ride starting at X, call the other stop Y. Assign more granular locations beyond X and Y. Select a random location (pair of coordinates) such that the location:

[1] Standard deviation calculated from historical data, 2017-2019.

    a. is at most a twenty minute[2] walk from the respective stop, and at least a five minute[3] walk.
    b. is not nearer any other stop A than it is to X, assuming A and X both service Y (and vice-versa, for Y).

  Call these coordinates C_X and C_Y.

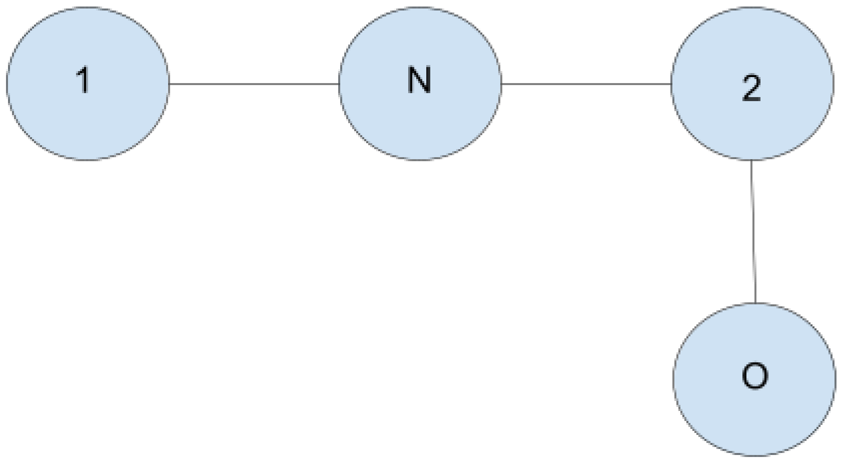
Now, with that process of granularity established, consider what the simulation should do when a stop is removed:

1. User selects a valid stop to remove from the MBTA. Call this stop X. Generate granular locations for all rides involving X.
2. All rides which include X must be re-routed. Find stop Z, which is connected to Y and is nearest to C_X. Nearest is defined as the shortest total transit time.

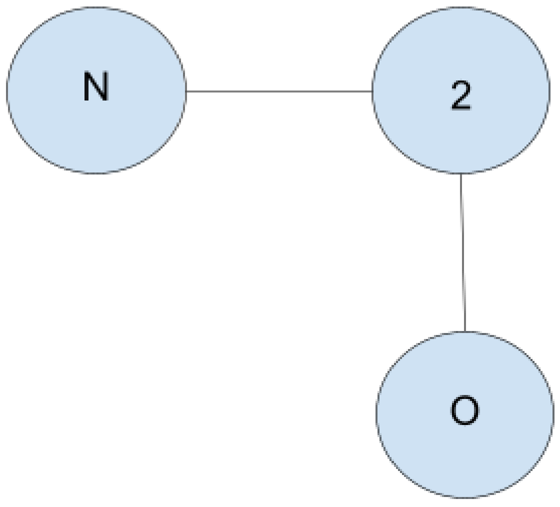
When a stop is added:

1. User inputs a stop name, coordinates, neighboring stops, and a valid line to add the new stop to. Call this stop N.
2. Find the nearest stops to N (which N may poach riders from). Call these stops P_i such that for all P_i:
    a. N is less than a 40 minute walk away. This means it could be possible, depending on destination, that N is a reasonable walking distance from the ride's granular location.
3. Now, for all P_i find all rides that include at least one P_i. Generate granular locations for them.
4. Then, from this set of rides, find all poachable riders, R_i such that for each:
    a. The other end of their route, O, is serviceable by N
    b. Using N is more efficient
        i. Calculate walk time to N and walk time to P_i
        ii. Calculate transit time for both.
            1. If N is between two existing stops:

[2] Twenty minutes was selected as this is commonly accepted as a "reasonable" amount of time to walk to reach a nearby train station.
[3] I assume here that there is a floor to how close any destination/origin could truly be to the station.



   a. Calculate the transit ride time from 1 to O (or vice versa depending on ride)
   b. Calculate the transit ride time from 2 to O (or vice versa)
   c. Compute the difference. Add to the transit ride time from 2 to O a fraction of the difference based on how close N is to 2 vs. 1.
2. If N is expanding a line:



   a. Calculate the transit ride time from 2 to O (or vice versa)
   b. Add based on the distance from N to 2
3. For both of these, depending on line, there is a speed assigned based on the average speed of those trains. Sourced from transitmatters.org.

        iii. Sum walk and transit time. The lesser of the two sums is the more efficient route; select that station.

5. For all R_i, replace P_i with N.

At this point, ridership metrics can be recalculated. I'd like to note that in practice, a flaw was noticed, and left in due to time constraints. In some instances, due to granularity, adding a stop would impact non-neighboring stops in perplexing ways. For instance, a rider would be poached from a stop multiple stops away, which at first blush makes no sense. Of course, under the right circumstances this makes complete sense – the rider's granularity is closer to the newly added stop, and/or their train is going in the direction of the newly added stop.

For instance, I've observed a new stop added between Wellington and Assembly on the Orange line poaching riders from Lechmere, likely due to a "perfect storm" of the riders' granular locations being near enough the new stop, and the riders likely heading to transfer from the Green line to the Orange line anyway to head north. The core problem is that the granular location calculation doesn't take into account whether or not the original ride is still the "optimal ride" for the granular locations determined. Going back to the example with Lechmere, if the algorithm factors in that the ride is headed north on the Orange line, it would avoid assigning a granular location near enough to the nearby Orange line stations for them to be faster. But, as it currently stands, the algorithm is agnostic to the bigger picture of the ride during granularization. A fix like this would likely decrease outlier results like these.

It's also worth noting that this model and simulation is more or less agnostic to transfer rides vs. non-transfers rides, as they generate no additional entry/exit data and are effectively the same as a non-transfer.

IV. Reflections

First of all, wow. None of this went how I thought. Everything took more time than it was planned to, reached a lower standard of quality than I expected, and required more effort throughout to get it there. I'm lucky I was able to get my top priority features done, because I sure didn't get to any of the other ones.

That being said, I think this project has come a long way. The simulation works, though it is flawed. Something I've come to appreciate is how iterative development and problem solving can be. It's impossible to devise a solution from the drawing board; it's necessary to try it, to fail, and to try again. Although Yoda once said "do or do not do; there is no try", he also said "greatest teacher, failure is".

And there were certainly failures on this project. For one, my model training scheme. It took a very, very long time to run and I’m not sure anything came of it that was productive. Every full size model that was mutated or crossed-over had the same fitness score, which was vastly lower than the varied fitness scores of the initially generated models. Although I’m pleased I was able to generate models and use them in my simulation, this was a big question mark on the overall project, which I simply couldn’t dedicate time to. I had invested too much in the model already and had to focus on the simulation.

For another, I had to scrap bus routes. This was disappointing; bus routes are integral to understanding the use of public transit. They link up the subway system and, although less frequently used than the subway, cover much wider terrain. Their usage would make the simulation more accurate. And there’s lots of opportunities for testing with bus routes, too; many more stops have been removed and added over the years to bus routes than the subway. Furthermore, there’s more applicability with simulating bus route modifications, as they are far more practical. If I could go back, I might rather focus on bus routes simply for that fact. Although it’s neat to do this with subway lines, the cost of adding or removing a subway station is immense. Bus stops, on the other hand, are little more than a signpost.

Something that became apparent to me as I edited my Schedule section was how heavily front-loaded this project truly was, in a deceptive way. While in the beginning, there was little concrete work to be done, the efforts required to flesh out the project were massive, and had many snags. In the end, when all that’s left to do is code it, there weren’t nearly as many hang-ups of the same scope.

On that note, is the issue of scope. There were a lot of bonus features planned for this project, all of which made it into the Future Work section below, and none of which were completed. Which is totally fine, of course. But I’m reminded on this project, as on all my projects, just how hard it is to manage a realistic scope. I’m pretty happy with how this scope was managed, even though I didn’t accomplish my personal goal of getting some of those bonus features into the project, because my top level scope for mandatory features was quite reasonable, achievable, and I did complete it.

This was my first project working with data, and I hope it won’t be my last. But by golly, is real-world data hard to work with. There’s inconsistencies, it isn’t the right shape, it’s not detailed enough, it’s not accurate enough, it’s too big, yet there’s never enough of it.

V. Testing

I've come up with a way to test my model and simulation's accuracy. There are real-world examples of that which I am simulating. Across the years, there have been new stops added to and removed from the subway and bus systems. This enables me to create a model with data from a year, then simulate a change made the following year, then compare with the data from the subsequent year. Bus changes are quite frequent, and there are plenty of examples which fall within the range of data I have (e.g. opening Airport for the Silver Line in 2018).

As for the subway: it's fortuitous that there was a big change to the subway so recently (GLX in 2022/2023). It's un-fortuitous that my subway data ends in 2019. The next closest changes to the subway are on GLB (Amory St. and Babcock St. in 2021), Orange Line (Assembly in 2014). Subway closures are infrequent too, with the most recent one being North Station and Haymarket (although only closing to GLC service in 2021). No other closures or openings were close to the data range I currently have – Red Line hasn't changed since the 80's, and Blue Line since the 50's.

There has been, however, a helping hand. The MBTA sent me data from this past year, enabling the GLX testing to occur. Unfortunately, I've plum run out of time to finish off the testing. I've created the model simulating the Green Line Extension, but there's substantial work remaining to actually build the infrastructure to compare this new data to a simulated model. The new data is aggregated across thirty-minute blocks, not the standard time blocks that the rest of my data is built on. Then there's the matter of how to actually compare the two datasets. Average percent change across all stations/time blocks would probably work. It's feasible, just not in time for my project's completion.

VI. Future Work

The first order of business for future work should be to complete the GLX testing I began during the project in earnest, as mentioned in the above section.

Following that, there's many other exciting avenues for future development in this project. The major standout is improvements to the model. The model itself is quite simple, and had major flaws during training which could not be fixed due to time constraints. A more accurate model would surely lead to a more accurate overall product.

At the outset of this project, I had a great desire to produce an interactable display on a website, with a map for point-and-click usability of the simulation. This was a medium-priority feature that fell by the wayside due to lack of time, but would really be great for quality-of-life purposes.

Furthermore, the simulation could be improved too; there's many factors which influence movement that are not accounted for. A standout is the station granulation functionality, which infers rider position relative to the known station using a random distribution combined with a min/max distance. Any point of interest model which takes into account what business surround the stations, their operating/busy hours, residential density, working hours for residents, etc. would have greater accuracy. This is such a rich topic it could merit an entire project in and of itself.

Another interesting development for the simulation would be to add the ability to, when adding a station, have it transfer to another line. Far more information would need to be taken from the user, including the line to transfer to, and other neighbor(s) on this secondary line. It would also be nice to be able to add an entirely new line. This is another relatively low-cost development that would be pretty nifty for simulation practicality.

Adding bus routes to the model and simulation would also have a great effect. Creating the map of bus routes would surely be time consuming due to the volume of stations, but otherwise connecting the bus routes should be a relatively painless process, and it would really boost the realism of the whole product.

For station subtraction, which in essence is simulating the outage of a subway station, it would behoove simulation users to be able to test the impacts of an outage against only a subset of riders, filtering by time. In other words, a feature enabling users to select a time range for their outage, which would empower users to compare the impacts on riders' experiences depending on when the outage was scheduled.

There's also the matter of new riders; in other words, people who start using transit after a station opens near them. Or conversely, those who stop using transit due to a station closing. This doesn't come up in my work at all, but it's something I thought about a lot. I am positive the recent Green Line Extension project increased total ridership due to the newfound convenience for those living in Medford and Somerville.

Finally, there's many other impacts changing a route has; congestion, cost, environmental impact, flow (to name a few). The model presented in this paper and the simulation built for it take none of these into account.

VII. References and Resources

1. https://ethz.ch/content/dam/ethz/special-interest/baug/ivt/ivt-dam/vpl/reports/1101-1200/ab1164.pdf
    a. Overview for the problem space.
2. https://ada.liacs.leidenuniv.nl/papers/BarEtAl14.pdf
    a. An example approach for using machine learning to tackle a problem similar to mine.
3. https://www.researchgate.net/publication/4156249_Activity_Recognition_and_Abnormality_Detection_with_the_Switching_Hidden_Semi-Markov_Model
    a. Another example of the above.
4. https://openproceedings.org/2011/conf/edbt/YanCPSA11.pdf
    a. Another example of the above.
5. https://www.researchgate.net/publication/341371936_Literature_Review_on_Genetic_Algorithm
    a. Overview of genetic algorithms. More specific to my approach.
6. https://mbtaviz.github.io/
    a. This was my primary inspiration. I'd like to one day have an interactable map like they do for my simulation.
7. https://dashboard.transitmatters.org/
    a. A valuable resource for transit data. I sourced train speed information from here.
8. https://www.mbta.com/developers/v3-api
    a. Reference for the API for MBTA. I didn't wind up needing the API itself for my project.
9. https://mbta-massdot.opendata.arcgis.com/
    a. This was where I found the freely available online MBTA ridership data.
10. https://mbta-massdot.opendata.arcgis.com/documents/MassDOT::mbta-system-data-guide/explore
    a. Useful for understanding how the MBTA data is organized.

VIII. Schedule

I've included my original schedule to show the outline of what my goals were, what I accomplished, and the struggles along the way. It is quite stream-of-consciousness. I consider this almost an appendix to the write-up above. Reading it walks through the project's progress, including the major impediments, and shows the notes I deemed important to chart the progression, prior to the code on GitHub speaking for itself.

November 30th: Alpha Version of Project

By this date, I aim to have a proof-of-concept working of my model, predicting paths. Very basic version of my primary A feature, perhaps on a smaller amount of data – maybe just a month's worth from 2023, like the visualizer, as a start.

Actual: I had completed a basic version of an individual of the model, without mutation, crossover, or fitness calculations to enable comparison. Model was not scaled up whatsoever.

December 20th: Catching Up

By this date, I aim to have completed my alpha version of the project as originally outlined in the November 30th deadline. Additionally, I will have contacted an ML professor at Tufts, requesting feedback on my model designs. To prepare for the next phase, I will have fleshed out my algorithm for simulation in pseudo-code.

Actual: I sent emails(12/8), but received no responses. I completed the code for the alpha version of the project, but was unable to complete training due to egregious inefficiencies in the code precluding me from generating even one full-size model. In addition to improving the performance of my code, I think the route forward will include crafting scale models and modifying the fitness calculator to account. Creating the smallest scale model (1/77300, with eight rail rides and one bus ride) took 4.5s. By this, I can estimate that a ¼ scale model might take 24 hours, ½ might take 48 hours, and a full-scale model could take 96 hours. Mind, this is just to generate a single model pseudo-randomly – to say nothing of the process of fitting, crossover, mutation, and new individual production to come with each generation of training. Batch creating models could be one avenue for code performance improvement. For this meeting, I have also attempted to generate a 1/60 scale model which by my calculations should take about 1.61 hours. In reality, it took closer to 1.17 hours, clocking in at 1:10:50. Worth noting, that running the model overnight cost around $120 (of my free trial credits). Cost wise, this will never scale. I can't do a request on every ride I generate. I could generate a few big queries which get all the distance answers and store them in data locally, to avoid future API calls. Given ~153 rail stops, and ~8,000 bus stops, this would be 64,023,409 distances to store. I could never do all those calls individually, but I can bundle multiple origins and destinations into a single API call. I'll have to see what the limit is on this, but if I can put as many as I want, conceivably this could only cost two API calls. Loading the returned data is another problem entirely – it will be enormous. I fleshed out the simulation algorithm in pseudo-code, but many challenges still remain.

January 31st: Beta Version of Project

By this date, I aim to resolve a few key hangups. For one, runtime on my current project. Either Pypy, a scale model, or something. What's the bottleneck? The Google cloud problem also needs a resolution. Can I get credits from Tufts? Can I work-around with my own model of MBTA? Can a bundled API call work? I need to get in touch with ML folks for a meeting; if they don't respond to emails, start knocking on doors. Finally, I'll need to continue contact with the OPMI to see if I can get any data. The answers from these investigations should leave me with enough information to restructure my goals and determine the further direction of this project.

Actual: I reached out regarding credits towards Google Maps API, but received a response saying that I should talk to Richard about it so…full circle, I suppose. I tried to bundle API calls together, but received errors from Google for putting too many elements in my query. I could try breaking it down and continuing the caching scheme but I'd need to play around a bit to figure out where the limit is.

I continued contact with OPMI. They told me they have faregate APC entries and exits by half-hour period at gated stations, processed from raw faregate data daily. They told me due to quality control reasons, most of the data is processed and aggregated, and that which I found on the ODP is probably the "best". I presume they mean "accurate"; for my purposes, however, I'd rather more granular data which is less accurate, as there's more inferences we can draw about the individual day, and I'm willing to sacrifice inferences we can draw about typical behavior. I responded requesting more information on this data, describing my project, and asking a few questions:

1. What, if any, data does the MBTA collect from stations without faregates? For instance, many of the Green Line stations have riders tap on the train or validate at the station. Is data collection here similar to that of the bus?
2. With respect to the faregate APC data, what sort of quality control does it usually undergo? I'm curious, so I can know what to look out for.
3. Ideally, I'd like to be able to get ridership data from a month (say, October 2023) with entries/exits timestamped (aggregated by half-hour period is good too), for all available stations. To what extent is this feasible? I understand if data limitations may preclude the bus from being included at the level of detail as the subway.

I received no response, after multiple emails, until 1/30, when they sent me a link to Dropbox, where there was some data. November 1st-15th, entries and exits for all stops on rapid transit (no bus), at each half hour. Far, far superior data to what I had.

I reached out to ML folks, and got a response back from Liping Liu. We met on 1/10. This meeting was very helpful; in short, it validated that the work I was doing was a proper approach to the problem, despite my reservations and lack of ML experience. Prof. Liu told me about some work he had done in the past on a similar problem, with birds. He had data reporting observations of bird species at different locations, at certain times, and was tasked with inferring the trajectory these birds had flown. He recommended a Monte Carlo Markov Chain, but agreed that a genetic algorithm would also be a suitable approach, and would likely yield similar results. As for the problem of the Google Maps API, Prof. Liu recommended building the subway map myself. I could use timetables for the subway/bus, calculate the differences, and create a weighted graph. I began the work but quickly ran into trouble. I desired to have all the stations and their neighbors neatly organized, but no such dataset exists. So I began the task of doing it by hand, but soon context switched to continue my work, as I wasn't sure this would be the best avenue. Prof. Liu also recommended the Data Lab @ Tufts, saying they may have relevant data. If I'm going to continue on the data driven route, and still want better data, I still need to dig deeper into what's at my disposal here. He also said I could reach out to them about Google Maps API credits, but they make no mention of it online; again, it may be moot at this point as my goals shift.

My brother gave me the good idea to reach out to an Urban Planning professor. I met with Shan Jiang, whose field neatly overlaps with my research on transit systems with big data. Prof. Jiang discussed with me other means for expanding my simulation's effectiveness, such as through cell phone data and census data. Cell phone tracking data can help place individuals throughout the day, and through pattern recognition cell phone users utilizing transit can be identified. Their trips can be inferred, as can their final destination/origins beyond the transit stops. Census data, which is far more accessible, can help in placing likely origin points for riders, while business data on busy hours can help in placing likely destination points for riders; this of course operates on the assumption that people generally travel to and from their residence.

Prof. Jiang also provided me with some MIT course slides from a Public Transportation Systems course to help me build a foundation on my understanding and the available literature. Finally, Prof. Jiang recommended an excellent paper to me called *Estimating a Rail Passenger Trip Origin-Destination Matrix Using Automatic Data Collection Systems* by Jinhua Zhao, et. al. Although all the background was interesting and relevant, the most pertinent element of the research was an inference model Zhao conducted on Chicago's CTA. In this model, they inferred passenger trips by leveraging a passenger ID associated with transactions conducted through the farecard medium. In short, they used the origin of subsequent trips to infer the destination of previous trips, and assumed every passenger started and ended at the same place each day. Zhao also referenced a few papers I found interesting, as they further discussed other methods of the same inference. I am still working my way through these adjacent papers and have little more to say about them at this juncture. After reading Zhao, I inquired to the OPMI if I may be able to access a similar model of data, with our Charlie Card system, but I'm not sure if they'll allow me. If so, it would be possible to recreate the model described by Zhao in Boston.

Reading these papers gave me a few other ideas for next steps; I could continue with my model, and compare it to existing ones, or I could use these models as the basis for an assumption of *some* model, and focus further efforts on my simulation with a toy example. This eliminated much of my data problems and would allow me to focus on the building of an algorithm with a perfect dataset of my own design. Of course, this would limit the applicability of my findings, but I think it would enable me to explore more deeply the ideas I have without the shackles of reality and data imperfections stymying my work. I could also just continue as I have been with the full project, but with each day that passes I fear the scope is too broad, and the cliffs too high.

With all this in mind, I consider amending the mission of my project. Perhaps I should shift from analyzing data, to generating data on a smaller scale to run a more modest simulation. Rather than focusing on the MBTA, I could build a toy example to demonstrate both my trajectory inference and my route modification simulation. The key question if this is to be pursued, is what data I should be generating? The more detailed the data, the more work that needs to be done in generation, but less in analytics – and vice versa. Additionally, what is lost by abandoning the real world? Do we care?

February 28th: Testing the Simulation

By this date, I aim to determine the bottleneck in my code. Likely the API calls? I will build a map of the MBTA to avoid API calls, produce an actual model that doesn't take a billion hours to generate, and work on the simulation, with a goal of getting some code written.

Actual: I've elected to to use time estimates for segments from google maps, to build a graph. This has been quite time consuming, but will save time in the long run. Recall that originally, creating the smallest scale model (1/77300, with eight rail rides and one bus ride) took 4.5s. On the new version, only 1.3s, over 70% faster! For this meeting, I have also attempted to generate a 1/60 scale model, this time taking only 00:04:09. I've also generated a full-scale model, taking 3:51:18, a huge leap forward in speed. However, the file is 191 MB, exceeding GitHub's limit of 100 MB files. I can probably fix this with compression or GitHub's LFS.

Interacting with all this data has driven me mad. The naming system for the stop_id value is absolute balderdash. I get that it's hard; I mean, airport naming codes/callsigns are the notorious example. But the scale here is but a fraction of that, and the name space is 2 characters more, which gives us over 11.5 million more options than IATA…how did the MBTA screw it up this bad? Yes, I'll elaborate: Forest Hills -> place-forhl, ok. North -> place-north, great, makes sense. Haymarket -> place-haecl? Oak Grove -> place-ogmnl? Community College -> place-ccmnl? place-hsmnl for Heath Street is just a crime…HEATH IS FIVE LETTERS!!! Massachusetts Ave -> place-masta? We shouldn't be abbreviating the literal words "station" or "stop" in the stop_id EVER. These are *all* stations/stops! But if we're going to, can we just pick a way to abbreviate station/stop and commit to it???

There's also a problem of the MBTA's poorly named stations – similar stations Chestnut Hill (D) and Chestnut Hill Ave. (B) get place-chhil and place-chill, respectively. What?!? Surely this is as confusing as it gets, the reader thinks; I raise the reader the identically named Saint Paul Street (B) and Saint Paul Street (C), which at the very least MBTA was thoughtful enough to give the names place-stplb and place-stpul (what, was place-stplc taken? (it isn't, I checked)), respectively. For what it's worth, that station's name got changed recently, too, which is another sack of beans I'll save for another rant.

Unfortunately, it gets worse: Ruggles -> place-rugg and JFK/UMass -> place-jfk; I mean, can we at least stick to a consistent length? I get these were never intended for human consumption, but they're downright laughable, and for something that is likely impossible to ever fix without a ton of tedious overhaul work, you'd think the person(s) in charge would've put a bit more thought into this.

March 27th:

By this date, I aim to have my model fully trained and simulation code written.

Actual: I managed to generate enough models to begin the training process, but due to the amount of time this consumed, I wasn't able to train them. As for the simulation, the code is all written but needs to be tested on a scale model to work out the bugs.

April 24th:

By this date, I aim to have my model fully trained and my simulation code functional on a full-size model, thus completing my two primary goal features. Any code written past this point should be for bonus features, quality-of-life, tests, examples, etc.

Actual: I did it!

May 8th:

By this date, I aim to finish writing my README and this write-up.